Lab in Control theory TSRT09+TSRT06
Multivariable control

This version: February 18, 2023

$$\begin{tikzcd}
 & & y_1 & & \\
G_{11} \arrow[rr] & & \Sigma \arrow[u] & & G_{12} \arrow[ll] \\
 & u_1 \arrow[ul] \arrow[dl] & & u_2 \arrow[ur] \arrow[dr] & \\
G_{21} \arrow[rr] & & \Sigma \arrow[d] & & G_{22} \arrow[ll] \\
 & & y_2 & &
\end{tikzcd}$$

REGLERTEKNIK
AUTOMATIC CONTROL
LINKÖPING

Name:	
P-number:	
Date:	
Passed:	

Chapter 1

Introduction

The purpose of this lab is to give an introduction to multivariable systems and design of controllers for multivariable systems. The process consists of two connect DC motors. We will start by familiarize us with the system, design PID controllers, and finally use state-space theory to design a controller using an observer and LQ-feedback. During the lab, we use MATLAB and SIMULINK for design, analysis and implementation. The developed controllers are implemented on hardware using National Instruments, allowing us to design the controller in SIMULINK and then compiling these to dedicated hardware. The computer on which the controllers are run are called xPC Target in the material.

Chapter 2

Introductory theory

This chapter gives a brief introduction to some theoretical aspects of the lab. For a more thorough description, the reader is referred to the course book.

2.1 Continuous systems - sampled systems

Since we use a computer to control the system, and thus use samples from a continuous process, we will shortly describe how this is handled.

Consider the system in 2.1. We see the system with the sampled in- and output signals $u(t_i)$ and $y(t_i)$. The block *Regleralgoritm* corresponds to a controller implemented in the xPC Target computer.

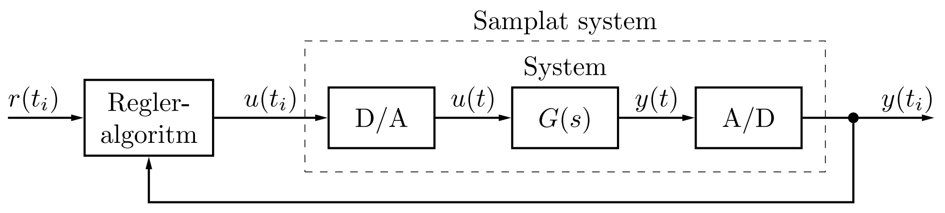


Figure 2.1: A sampled system.

If the system is sampled sufficiently fast, the sampled signals are a good approximation of the continuous signals, and the sampling will only marginally reduce performance and robustness. With a good choice of sampling time we can use an approximation (discretization) of a continuous controller using, e.g., Euler or Tustins approximation, see Chapter

11 in the basic control course book. In this lab, we will not investigate issues from sampling, but simply assume that we are using a sampling time which is short enough for us to use the sampled continuous-time controller without any changes.

2.2 Controller structure

The controllers we use in the lab can be seen as two parts, one feedforward term $F_r(s)$, and one feedback $F_y(s)$. The input to the system is thus

$$u(t) = F_r(p)r(t) - F_y(p)y(t)$$

see Figure 2.2. In the PID-part of the lab, we will only use controllers of the form $F(s) = F_r(s) = F_y(s)$. In this case, the controller acts as a compensator in front of $G(s)$ with input $r(t) - y(t)$. In the LQ-part, $F_y(s)$ och $F_r(s)$ will be different.

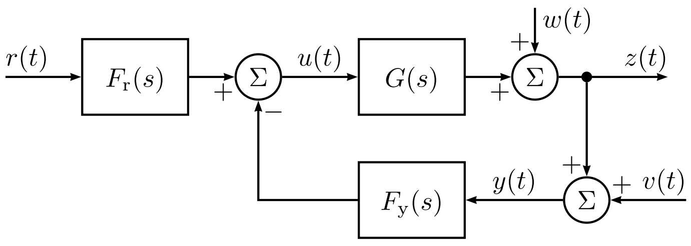


Figure 2.2: System and controller

2.3 Goal

The purpose of the controllers is to create a design which allows us to control the output y_1 och y_2 and have them follow r_1 and r_2. This is called the servo problem. We would like r_1 to only affect y_1 and not y_2, and r_2 only affect y_2. This is called decoupling. Decoupling of the closed-loop system $G_c(s)$ is achieved if the loop-gain $G(s)F_y(s)$ is diagonal. For static decoupling we only require $G(0)F_y(0)$ to be diagonal.

The controller should of course be robust against modeling errors, and insensitive to process noise and measurement errors.

2.4 Theory

How should $F(s)$ be selected? In the first part of the lab we use a diagonal controller where each diagonal element of $F(s)$ is a PID-controller. Using a diagonal $F(s)$ (or permutation of a diagonal) means that we are controlling the system as if it was composed of several independent parallel systems without any coupling. Any actual coupling is seen as a disturbance.

Following from there, we use controllers which use all degrees of freedom, i.e., we let $F(s)$ be a full matrix. This means that the controller will be able to exploit the fact that $G(s)$ is multivariable and coupled. To achieve decoupling, we essentially try to design controllers $F(s)$ which somehow approximates $G^{-1}(s)$.

As a final experiment, we develop an LQ controller, and since we cannot measure all the states, we need an observer. Given a state-space description of the system

$$\dot{x}(t) = Ax(t) + Bu(t) + Nv_1(t) \tag{2.1}$$

$$y(t) = Cx(t) + Du(t) + v_2(t) \tag{2.2}$$

$$z(t) = Mx(t) \tag{2.3}$$

where $v_1(t)$ and $v_2(t)$ are unit disturbances (white noise) with intensities (covariance) R_1 and R_2. The cross-spectra between $v_1(t)$ and $v_2(t)$ is assumed to be zero. We design a state-feedback

$$u(t) = -Lx(t). \tag{2.4}$$

We typically want a linear combination of states ($z = Mx$) to quickly approach zero after a disturbance. Additionally, we have penalties on control inputs. In LQ-theory, we select the weight (penalty) matrices Q_1 och Q_2 in

$$J(Q_1, Q_2) = \int_0^\infty \left(z^T(t)Q_1z(t) + u^T(t)Q_2u(t)\right) dt. \tag{2.5}$$

The matrices Q_1 and Q_2 reflect the desired compromise between closed-loop performance and control usage. The state-feedback L in (2.4) minimizing (2.5) is found in Matlab with

```
L = lqr( A, B, M'*Q1*M, Q2 ).                (2.6)
```

Since we cannot measure all states (we only measure angles), we have to estimate them using an observer

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t) - Du(t)). \tag{2.7}$$

The control input is then calculated using the estimated states instead

$$u(t) = -L\hat{x}(t). \tag{2.8}$$

A Kalman filter is an observer where the observer gain K is designed so that the estimation error covariance is minimized. With this, an optimal compromise between performance and sensitivity, with respect to the noise model, is achieved. The Kalman filter is thus defined by the K minimizing

$$P = E\left[\tilde{x}(t)\tilde{x}^T(t)\right], \tag{2.9}$$

where

$$\tilde{x}(t) = x(t) - \hat{x}(t) \tag{2.10}$$

and $\hat{x}(t)$ is given by (2.7). The optimal gain K is obtained in MATLAB with

```
K = lqe( A, N, C, R1, R2 ).                    (2.11)
```

The strategy to combine the LQ-feedback and the Kalman filter state estimate is called Linear Quadratic Gaussian (LQG).

Chapter 3

Equipment

3.1 Computer and software

We use Windows computers with MATLAB and SIMULINK. In MATLAB we use standard commands from *Control System Toolbox* to design and analyze controllers.

SIMULINK is used for analysis, design and implementation of the controllers. A number of SIMULINK-files are prepared for the lab

- *pid_xpc.mdl*, SIMULINK-model for PID-control of the system through xPC Target.
- *lq_xpc.mdl*, SIMULINK-model for LQ-control of the system through xPC Target.
- *lq_model.mdl*, SIMULINK-model over the system controlled using LQ. Used only for simulation.

The two models of the DC-motors described in Chapter 3.2 are available in state-space form (`As`, `Bs`, `Cs`, `Ds`) in the files *sys1.mat* och *sys2.mat*. A couple of MATLAB-files are also available

- *pid_params.m*, MATLAB-script to define PID-parameters. Updates the controller in *pid_xpc.mdl*.

- *discretize_observer.m*, MATLAB-script to create a discretized observer used by *lq_xpc.mdl*.

3.2 The system

The system we control in the lab consists of two DC-motors, connected using a mysterious gray box. Through a switch on the gray box, the dynamics can be changed. In the lab, we will work with the system when the switch is in position 1 (position 2 only if you have time).

The measured outputs of the system are the angles of the two DC-motors while inputs are voltages sent to the gray box. Figure 3.1 on page 9 illustrates the complete setup.

The system $G(s)$ can be described approximately as

$$G(s) = \begin{pmatrix} \frac{48}{s(0.25s+1)} & \frac{96}{s(0.25s+1)} \\ \frac{96}{s(0.25s+1)} & \frac{48}{s(0.25s+1)} \end{pmatrix} \tag{3.1}$$

when the switch is in position 1. Notice the coupled, but simple symmetric dynamics.

In position 2, it is given by

$$G(s) = \begin{pmatrix} \frac{48}{s(0.25s+1)} & \frac{96s+96}{s(0.25s+1)^2} \\ \frac{96s+96}{s(0.25s+1)^2} & \frac{48}{s(0.25s+1)} \end{pmatrix} \tag{3.2}$$

State-space models

A minimal state-space representation of $G(s)$ is

(position 1)

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -4 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 \\ 192 & 384 \\ 0 & 0 \\ 384 & 192 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

(position 2)

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -4 & 1 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & -4 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 \\ 192 & -1536 \\ 0 & -4608 \\ 0 & 0 \\ 1536 & 192 \\ -4608 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Offset-adjustment of motors
Both DC-motors have a potentiometer to adjust the voltage offset. Before you start experimenting, adjust these so the motors stand still when the equipment is turned on but the input to the system is zero.

Control limitations
The linear models above are of course only approximations. In particular, they are not good when the inputs to the system is larger than $5V$. A controller designed to give a fast system naturally leads to large inputs. This fact gives us a physical limitation on the performance of the system, if we only want to control the system in regions where our models are good. If we go outside this region, we can no longer guarantee stability and performance, even though theory might guarantee this.

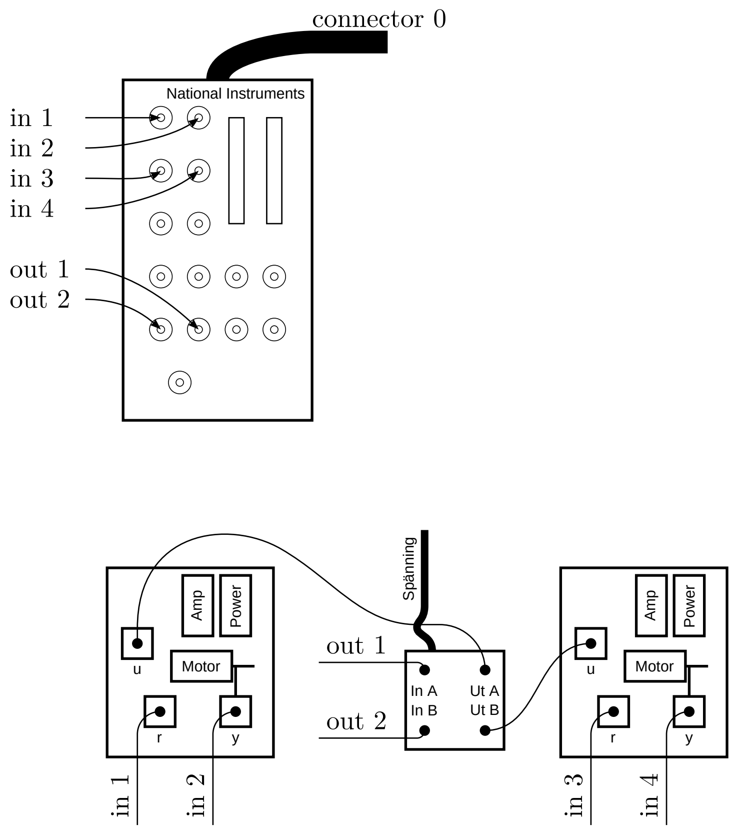


Figure 3.1: The two DC motors, the gray box and xPC Target via National Instruments. Note that Out1 and Out2 are flipped in order on connector 0. You should not have to touch any cable or care about this.

Chapter 4

Lab task

Getting started

- Make sure the system is setup according to Figure 3.1.
- Turn on power on all units (large red button on brown box, two small green buttons on each DC motor board). Note that some old bad controller might be downloaded to the DSP, so you might want to turn off the motors again until you have downloaded your own code (it will be very obvious if you want to turn off the motors...).
- Open `K:\TSRT09\labs` and copy the directory `\multivar_1_2` to, e.g., your student account. The copied directory might be write protected. Right-click the copied directory, Properties/Egenskaper, and make sure it is not marked as read-only
- Start MATLAB R2016b. (IMPORTANT! Use this specific version!)
- Go to the directory with the copied files
- Open the file `xpcsetup.m` in MATLAB and follow the steps in the file to initialize the communication between the desktop computer and the xPC Target computer. Note in particular what has to be done if several targets are listed.
- Make sure the switch on the gray box is in position 1
- Load the model, `load sys1`.
- Write `whos` in MATLAB to see the data you just loaded.

PID

Open the SIMULINK file *pid_xpc.mdl*. Define PID-parameters by editing and running `pid_params`. Only use P or PD-control for now, i.e., let $T_I = \infty$. We will come back to integral action later.

Compile the controller to the xPC Target computer with the command `slbuild( 'pid_xpc' )`. The compilation might crash the first time you run it, but if you run it again it should work.

Activate the controller with `tg.load( 'pid_xpc' )`. The controller can be started and stopped with `tg.start` and `tg.stop`.

When you change PID-parameters through `pid_params`, the values in the compiled controller will be updated, as long as you do not change the structure of the controller (i.e. turning on/off integral and derivative action) which requires a new compilation and activation. Similarly, any structural change in the SIMULINK model defining the controller requires recompilation and activation.

The SIMULINK file *pid_xpc.mdl* is initially setup for open-loop control. Study the model and make sure you understand why it implements open-loop control.

Open-loop control
To get a feeling for the system, we start with open-loop control (adjusted by the switches in the SIMULINK model), i.e., no measurement feedback and simply use $u = F_r r$.

Can you make both motors stand still, or easily control the angle of the two motors independently?

....

....

Control with diagonal controller
Close the first loop using a PID-controller. The PID controller with parameters defined in `pid_params` is $K(1 + \frac{1}{T_I s} + T_d \frac{s}{s/w+1})$. We disregard the second loop for the moment and let it be uncontrolled (send zero as input in the SIMULINK model). Compile and activate the model again (remember, after every structural change in the model, you have

to recompile).

The measurement discs sometimes get dislocated on the DC-motors, which means that the angle you read is different from the angle (or more precisely voltage) the computer actually measures. To see the measurements and control inputs in real-time, press *input* on the monitor, and you will see these signals on the DSP monitor.

Can you stabilize and achieve good reference tracking on the first loop? Which controller parameters are you using?

. . . .

. . . .

Close the second loop instead of the first and repeat the experiment, using the same PID-parameters, and confirm that it works equally well on that channel.

Now close both loops using your controllers. What happens?

. . . .

. . . .

Let us analyze this theoretically and compute the closed-loop poles. In *pid_params.m* you see that two transfer function `F1` and `F2` are defined. Use these to define your multivariable controller `F`. State-space models (`As`,`Bs`,`Cs`,`Ds`) for the system $G(s)$ were loaded earlier. Use these to create a model `G` with the command `ss`. The closed-loop system can now be obtained with the MATLAB command `feedback`. The command `feedback(N,M)` generates $(I + N(s)M(s))^{-1}N(s)$. The closed-loop system is thus created with

```
Gc = feedback(G*F,eye(2));
```

How many states/poles should the closed-loop system have? (How many states does the open-loop system have. How many states do you need to implement your two PID controllers? Note that the number of states in the controller depends on which components you actually use in the controller)

. . . .

. . . .

In some cases, numerical issues might lead to non-minimal realizations and thus to too many states. This can be solved with the command

```
Gc = minreal( Gc );
```

Always use this command when creating various transfer functions. The closed-loop poles, i.e., the eigenvalues of `Gc.a` in a minimal realization, can be computed with `pole(Gc)`. The number of poles should coincide with the number you gave above. What can you say about your poles?

. . . .

. . . .

Investigate which inputs and outputs that are most coupled, using RGA-analysis (see chapter 8 in the course book). When doing this, it is most easily done by simply looking at the model given in Section 3.2, and work with that. Note: since $G(s0$ has a pole in 0, you cannot evaluate G at $\omega = 0$, but you can use instead ω which is very small.

. . . .

. . . .

A basic technique for multivariable control is to pairwise control the most coupled input-output pairs. How can you change the connection between the controller and the system to achieve this? Do these changes in the controller definition *pid_xpc.mdl* (*not physically!*) and make corresponding changes in your variable `F`.

Test your new controller.

. . . .

. . . .

Is the closed-loop model stable? Check the poles.

. . . .

. . . .

Can we achieve decoupling with this type of controller? Hint: preparation question 2.

. . . .

. . . .

Now study the singular values of the loop gain $G(s)F(s)$ and of the closed-loop system $G_c(s) = (I + G(s)F(s))^{-1}G(s)F(s)$ using the command `sigma`.

Which typical properties can you see in the loop-gain (low-frequency gain, break-points, signs of multivariable and/or cross-coupling)? Can you connect the low-frequency gain and break-points with poles and zeros in the open-loop system and the controller?

. . . .

. . . .

With a small number of inputs and outputs, we can also look at single channel gains. Use `bodemag` to study amplitude gains on the closed-loop system. How can you see cross-coupling here?

. . . .

. . . .

We will now study how sensitive the system is for *input disturbances*. We can test this in practice by turning the *zero-offset*. By adjusting this, an additive, for the controller unknown, voltage is added to the control input that is delivered to the DC-motors. By definition, an input disturbance!

Start by turning off the controller, and adjust the *zero-offset* on both motors to make sure they stand still.

Now turn on the controller, and switch monitor so you can see your signals. Change the *zero-offset* significantly (introduce disturbance!) so you see large movements. Do the motors stop? (i.e., does the total voltage applied the motors go back to 0 statically)? Does the control error approach 0?

. . . .

. . . .

To analyze how the system reacts to input disturbances (which we call w_u) we can study the input sensitivity function $S_u(s) = (I + F(s)G(s))^{-1}$ (transfer function from a disturbance added to the input computed by the controller, to the input actually inserted in the system), and the transfer function from an input disturbance to the controlled output y. This transfer function is $G_{w_u y} = G(I + F(s)G(s))^{-1}$ which can be written as $(I + G(s)F(s))^{-1}G(s)$. These transfer function can be defined using `feedback`. Do not forget to use `minreal`.

Draw a block diagram with F, G, r, w_u and y and clearly indicate which transfer functions we are computing here.

Start by studying the singular values of the input sensitivity function $(I + F(s)G(s))^{-1}$. What will the static influence on the applied input be when we have a constant input disturbance? Is it consistent with experimental results?

. . . .

. . . .

Now study the singular values of the transfer function from input disturbance to the controlled variable y, and in particular the static influence. Is it consistent with experimental results?

. . . .

. . . .

The input applied to the system is $W_u(s) + F(s)(R(s) - Y(s))$. For the system to stand still this signal has to be 0 stationary when $w_u(t)$ is a constant. Can the control error $r(t) - y(t)$ be 0 stationary, with a constant input disturbance acting, if $F(s)$ is a PD controller?

. . . .

. . . .

Now change your controller to incorporate integral action, and perform the same experiment again. Also redraw the singular values. What happens now?

. . . .

. . . .

Adjust *zero-offset* back to zero again before you proceed (stop the controllers and adjust until nothing moves).

More advanced controllers
Let us now study the system we control and see if we can find a more complex control structure to achive decoupling. Modify the controller in the SIMULINK file according to your solution of preparatory exercise 4. (Practically this means H in the preparatory exercise is the gray box, and $\bar{G}(s)$ are the two DC motors.) Write `simulink` in MATLAB to make all SIMULINK blocks available. The SIMULINK blocks `Mux`, `Demux` and `Gain` might be useful to implement the controller in this task. Try your new controller. Note that a too fast closed-loop systems might lead to instability due to model errors.

Check the closed-loop poles.

. . . .

. . . .

Now study the open-loop gain and the closed-loop gain using `sigma` and `bodemag`, and compare with the previous control. Can you see any signs of model errors when you compare the experimental behavior with the theoretical gains? Also study the input sensitivity function.

. . . .

. . . .

Extra task (when you have finished the lab if you have time)

Repeat the control design for the system with the gray box switch in position 2. The model is available in the file *sys2.mat*.

LQ

We will now design LQG controllers. This gives us a simple approach to design multivariable controllers, without any special tricks to deal with coupling. We start with simulations to design the LQG-controller through tuning of the weights Q_1 and Q_2 in (2.5) until we obtain step responses with desired performance and reasonable control use. Our initial goal is to design a closed-loop system with bandwidth approximately $\omega_b = 2$ rad/s, while not using control inputs beyond the limit of 5 volts. When we have obtained a satisfactory controller we use it on the real system. Since we cannot measure all states (and those that we do measure might have measurement errors), an observer has to be used in practice.

Our system is written in the form

$$\begin{aligned} \dot{x} &= A_{\mathrm{s}}x + B_{\mathrm{s}}u + N_{\mathrm{s}}v_1 \\ y &= C_{\mathrm{s}}x + v_2 \end{aligned} \tag{4.1}$$

where v_1 and v_2 are white Gaussian noise with mean 0 and covariances R_1 och R_2. Unfortunately we do not know the actual values of R_1 och R_2 so these are also tuning variables.

Since we cannot measure all states we need an observer to estimate them before performing the LQ-feedback. The structure for an observer is setup in the SIMULINK models, but first we have to compute the observer gain K. In MATLAB we do this with

```
K = lqe( As, Ns, Cs, R1, R2 )
```

Since we do not know R_1 and R_2, we have to make an educated guess based upon what we know on their relative size, i.e, the relative amount of process vs measurement noise. In this application, a reasonable guess is that the measurement error is more problematic than the process error, hence:

$$\begin{aligned} R_1 &= \texttt{0.1 * eye(2)} \\ R_2 &= \texttt{eye(2)} \end{aligned}$$

Now we have to compute the state-feedback. In MATLAB:

```
L = lqr( As, Bs, Q1, Q2 )
```

where Q_1 is the weights on the states and Q_2 is the control input weight. By symmetry, it is reasonable to have the same penalty on both inputs,

$$Q_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and simply vary Q_1 as a design parameter (as it is the relative size of Q_1 and Q_2 which matters)

To achieve an identity static gain on the closed-loop system from reference r to controlled variables $z = Mx$ (which happens to be the same as the measurements in our setup), we have to feedforward the reference suitably. With $u = L_0 r - Lx$, the closed-loop system is given by

$$\begin{aligned} \dot{x} &= (A - BL)x + BL_0 r \\ z &= Mx \end{aligned}$$

The transfer function of the closed-loop is thus $M(sI-(A-BL))^{-1}BL_0$, and with the static gain requirement $G_c(0) = I$ the choice of L_0 can be derived. When we replace the states x with their estimates $\hat{x}$ from the observer, the arguments above will not change, as the transfer function from r to y does not depend on the observer.

Start by creating a state-feedback in which the angular velocities are not weighted. Weight both angles identically, and do not use any weight outside the diagonal, i.e., use Q_1 on the form

$Q_1 =$ `diag( [ qVinkel 0 qVinkel 0 ] )`

Use *lq_model.mdl* to simulate the system, and singular values of the closed-loop systems, for various Q_1, until it looks promising and satisfies the requirements. Use *lq_xpc.mdl* to test the controller in practice. Do not forget to run the file *discretize_observer.m* to discretize the continuous-time observer. Are the results satisfactory?

. . . .

Look at the closed-loop poles, i.e, eigenvalues of $A_\text{s} - B_\text{s}\, L$. Describe how they differ from the poles you obtained with the PID-based controllers

. . . .

Try to weight the angular velocities also. What happens with the coupling when you start penalizing the velocities? (this is not a generic feature of LQ but a phenomena on this system)

. . . .

Extra task

Repeat the control design for the system with the gray box switch in position 2. The model is available in the file *sys2.mat*.

Chapter 5

Preparation

Before the lab, there will be a small test. Related material is this pm, theory in the book, and the questions below. note that the test is done individually, without computer or book. To pass, you have to have 3 answers right out of 5.

For the LQ-part of the lab to be done in time, chapter 8.5 and 9 in the course book, and the theory part of this pm, must have been read.

1. Let $Y(s) = G(s)U(s)$ where,
$$G(s) = \begin{pmatrix} \frac{1}{s+1} & \frac{5}{s+2} \\ \frac{1}{s+1} & \frac{5}{s+1} \end{pmatrix}$$
compute poles and zeros of the system $G(s)$. Which singular values do you have at 3 rad/s? (Use MATLAB or a calculator when the expressions gets messy in the singular value computations)

2. Consider the system on figure 2.2 where $F_y(s) = F_r(s) = F(s)$. Show that decoupling in the closed-loop system $G_c(s)$ is achieved if the open-loop system $G(s)F(s)$ is diagonal. (A diagonal matrix has non-zero elements only on the diagonal).

3. Let $Y(s) = G(s)U(s)$ where,
$$G(s) = \begin{pmatrix} \frac{1}{s+1} & \frac{2s}{2s+1} \\ \frac{2s}{2s+1} & \frac{1}{s+1} \end{pmatrix}$$
Compute RGA for $G(0)$ and $G(\infty)$. can you make a qualified guess, based on your RGA analysis, if it is hard or easy to control this system using two single-variable PID controllers.

4. Start by showing that $G(s)$ (i.e., (3.1)) can be factorized in the following form:

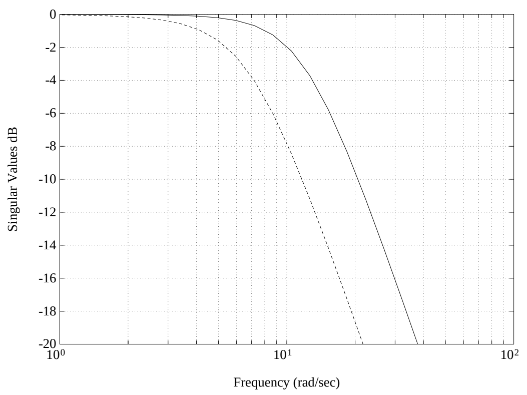
$$G(s) = \bar{G}(s)H = \begin{pmatrix} \bar{G}_1(s) & 0 \\ 0 & \bar{G}_2(s) \end{pmatrix} H$$

where H is a matrix independent of s. How can we design a controller $F(s)$ such that $G(s)F(s)$ is diagonal, i.e, decoupling is achieved? Let the controller $F(s)$ contain the PID-controller

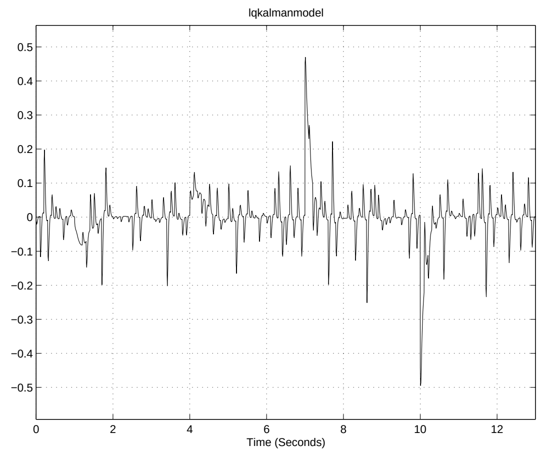
$$F^{\text{diag}}(s) = \begin{pmatrix} F_1(s) & 0 \\ 0 & F_2(s) \end{pmatrix}.$$

For instance, $F(s) = W_1 F^{\text{diag}}(s) W_2$, where W_1 and W_2 are the two matrices you have to compute.

5. Determine boundaries on the bandwidth when the closed-loop singular values are as below



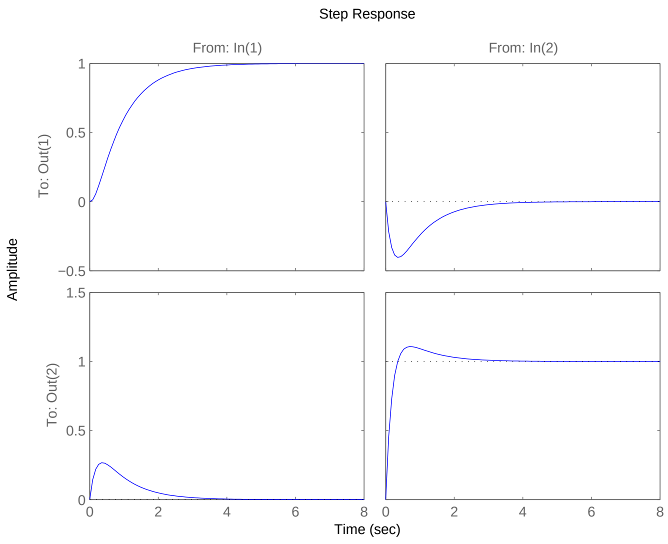
6. A control system with feedback from estimated states has been designed and looks promising in simulations. When the solution is tested in practice, the control input unfortunately looks as below. It is assumed that the problem is due to the observer. In which direction would the noise variances matrices R_1 och R_2 be adjusted to obtained a smoother input signal?



7. A system is controlled with state-feedback, computed in MATLAB using

```
L = lqr( A, B, [ 2 4 ; 4 9 ], [ 1 0.9 ; 0.9 1 ] )
```

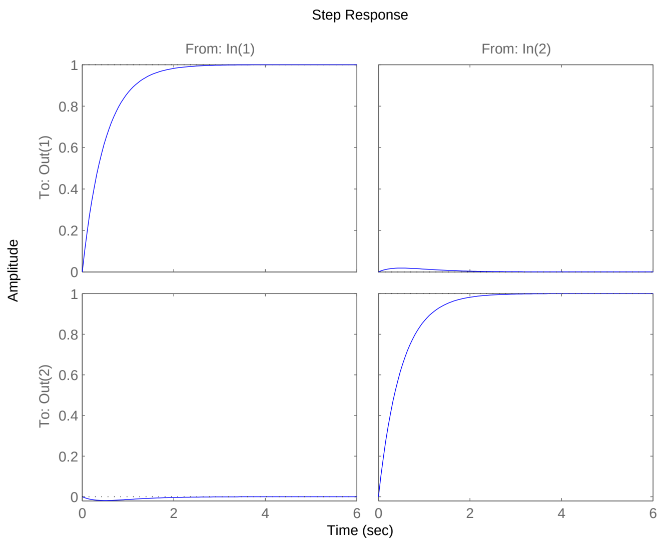
A step response is shown in the figure below. It does not look good, but is optimal in some sense. Which sense?



8. (Continued from previous question) With the simple choice

```
L = [ 1 0 ; 0 1 ]
```

the step response is shown in the figure below, which perhaps looks better than the result achieved using LQ in the previous question. What are the pro- and cons in choosing L directly, vs using LQ to design it?


Step Response
From: In(1)
From: In(2)
Amplitude
To: Out(1)
To: Out(2)
0
0.2
0.4
0.6
0.8
1
0
2
4
6
Time (sec)